ICP-7

Sreeja Reddy Konda

700756597

GITHUB: https://github.com/SreejaReddyKonda/Neural-Network-Sreeja/blob/main/ICP-7/ICP7.ipynb

Video: https://drive.google.com/file/d/1bnQy4ztXifiddkuwrk4CASC77JVhvzlZ/view?usp=sharing

1.

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import re

from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('/content/Sentiment (3).csv')
# Keeping only the neccessary columns
data = data[['text','sentiment']]

data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)

X = pad_sequences(X)

embed_dim = 128
lstm_out = 196
def createmodel():
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3,activation='softmax'))
```

```
embed_dim = 128
lstm_out = 196
def createmodel():
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
    return model
# print(model.summary())

labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33, random_state = 42)

batch_size = 32
model = createmodel()
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2)
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
print(score)
print(acc)
print(model.metrics_names)
```

```
291/291 - 48s - loss: 0.8208 - accuracy: 0.6428 - 48s/epoch - 166ms/step
144/144 - 4s - loss: 0.7668 - accuracy: 0.6614 - 4s/epoch - 31ms/step
0.7668231725692749
0.6614242196083069
['loss', 'accuracy']
```

```python
import tweepy
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import re

# Load the saved model
model = load_model("/content/sentiment_model.h5")

# Define a function for preprocessing text
def preprocess_text(text):
    text = text.lower()
    text = re.sub('[^a-zA-z0-9\s]', '', text)
    return text

# Example new text data
new_text = "A lot of good things are happening. We are respected again throughout the world, and that's a great thing. @realDonaldTrump"

# Preprocess the new text data
new_text = preprocess_text(new_text)

# Tokenize and pad the new text data
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts([new_text])
X_new = tokenizer.texts_to_sequences([new_text])
X_new = pad_sequences(X_new, maxlen=model.input_shape[1])

# Make predictions
predictions = model.predict(X_new)

# Determine the sentiment based on the prediction
sentiments = ['Negative', 'Neutral', 'Positive']
predicted_sentiment = sentiments[predictions.argmax()]

# Print the result
print("Predicted Sentiment: " + predicted_sentiment)
```

Explanation:

1.Import Libraries: It starts by importing necessary libraries. `tweepy` is used for accessing the Twitter API, `keras`. is used for building and loading the neural network model, and `re` for regular expression operations.

2.Load Pre-trained Model: The pre-trained sentiment analysis model is loaded from a saved file (`sentiment_model.h5`). This model is assumed to be trained to classify text into sentiments.

3.Preprocess Text: The `preprocess_text` function is defined to clean the input text by converting it to lowercase and removing non-alphanumeric characters. This ensures the model receives the text in the format it expects.

4.Example Text: A sample tweet is provided as `new_text`. This text is then preprocessed to remove unwanted characters and format it properly.

5.Tokenize and Pad the Text: The text is tokenized using Keras' `Tokenizer`, which converts the text into a sequence of integers where each integer represents a specific word in a dictionary. The sequence is then padded to ensure it has a fixed length, matching the model's input requirements.

6.Make Predictions: The preprocessed and formatted text is fed into the model to predict its sentiment. The model outputs a probability distribution across the possible sentiment classes (Negative, Neutral, Positive).

7.Determine Sentiment: The sentiment with the highest probability is selected as the predicted sentiment for the input text.

2.

2. Apply GridSearchCV on the source code provided in the class

```
[ ] !pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.4.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

```
[ ] from scikeras.wrappers import KerasClassifier
```

```
[ ] import pandas as pd
    import re
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
    from tensorflow.keras.utils import to_categorical
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.preprocessing import LabelEncoder
    from scikeras.wrappers import KerasClassifier

    # Assuming the data loading and preprocessing steps are the same

    max_features = 2000
    tokenizer = Tokenizer(num_words=max_features, split=' ')
    # Assuming tokenizer fitting and text preprocessing is done here
```

```
def createmodel(optimizer='adam'):
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

# Define the KerasClassifier with the build_fn as our model creation function
model = KerasClassifier(model=createmodel, verbose=2)

# Define hyperparameters to tune
param_grid = {
    'batch_size': [32, 64],
    'epochs': [1, 2],
    'optimizer': ['adam', 'rmsprop']
}

# Initialize GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)
# Fit GridSearchCV
grid_result = grid.fit(X_train, Y_train)

# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
194/194 - 37s - loss: 0.8596 - accuracy: 0.6328 - 37s/epoch - 192ms/step
97/97 - 2s - 2s/epoch - 23ms/step
194/194 - 41s - loss: 0.8563 - accuracy: 0.6297 - 41s/epoch - 210ms/step
97/97 - 3s - 3s/epoch - 34ms/step
194/194 - 36s - loss: 0.8773 - accuracy: 0.6278 - 36s/epoch - 186ms/step
97/97 - 2s - 2s/epoch - 23ms/step
194/194 - 32s - loss: 0.8712 - accuracy: 0.6326 - 32s/epoch - 167ms/step
97/97 - 3s - 3s/epoch - 28ms/step
```

Explanation:

1.Library Imports: It starts by importing necessary libraries. `pandas` for data manipulation, `re` for regular

expressions, `tensorflow.keras` for building and training the neural network model, `sklearn.model_selection` for splitting the dataset and conducting grid search, and `scikeras.wrappers` to wrap Keras models for use with scikitlearn.

2. Model Building Function: The `createmodel` function defines the architecture of the neural network using Keras' Sequential API. It includes an Embedding layer for text input, a SpatialDropout1D layer to reduce overfitting, an LSTM layer for learning from the sequence data, and a Dense output layer with a softmax activation function for classification. The optimizer for compiling the model can be adjusted, making the model flexible for hyperparameter tuning.

3. KerasClassifier Wrapper: A `KerasClassifier` wrapper is used to make the Keras model compatible with scikitlearn's

grid search functionality. This allows the use of scikit-learn's `GridSearchCV` for hyperparameter tuning.

4. Hyperparameter Tuning: A parameter grid is defined with different values for batch size, number of epochs, and optimizer type. `GridSearchCV` is then used to exhaustively search through the parameter grid for the best model configuration based on cross-validation performance. It evaluates model performance for each combination of parameters across a specified number of folds of the training data.

5. Model Training and Selection: `grid.fit(X_train, Y_train)` trains the model using the training data across all combinations of parameters specified in `param_grid`, using cross-validation. After fitting, it identifies the combination of parameters that resulted in the best model performance.

6. Results Summary: Finally, the best performance score and the hyperparameters that led to this best score are printed. This provides insights into which settings worked best for the given text classification task.